John von Neumann Institute for Computing

The HARWEST High Level Synthesis Flow to Design a Special-Purpose Architecture to Simulate the 3D Ising Model

Alessandro Marongiu, Paolo Palazzari

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 785-792, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

The HARWEST High Level Synthesis Flow to Design a Special-Purpose Architecture to Simulate the 3D Ising Model

Alessandro Marongiu[1,2] **and Paolo Palazzari**[1,2]

[1] Ylichron Srl
Via Anguillarese, 301
00123 S. Maria di Galeria (Rome), Italy
E-mail: {a.marongiu, p.palazzari}@ylichron.it

[2] ENEA - Computing and Modelling Unit
Via Anguillarese, 301
00123 S. Maria di Galeria (Rome), Italy

The 3D-Ising model is a mathematical model adopted in statistical mechanics, used to model basic phenomena as the paramagnetic-ferromagnetic transition. Due to its numerical intractability, many Monte Carlo methods have been proposed to solve the 3D Ising model. In this work we refer to the Heat-Bath algorithm and we show how it can be efficiently implemented onto an FPGA-based dedicated architecture through the adoption of the HARWEST High Level Synthesis (HLS) design flow, developed by Ylichron Srl.
Before going into the details of the specific implementation, the HARWEST HLS flow is reviewed, recalling the main modules and functionalities that are part of the flow.
The results achieved in 3D-Ising Model test case evidentiate speedup factors ranging from 1 to 3 orders of magnitudo with respect to optimized implementations on some current high-end processors.

A Introduction

The 3D-Ising model[1] is a mathematical model adopted in statistical mechanics, used to model basic phenomena as the paramagnetic-ferromagnetic transition. While the 2D Ising model admits an analytical solution, the more interesting 3D case can be solved only with computationally intensive numerical methods. Due to its numerical intractability[2], many Monte Carlo methods have been proposed to solve the 3D Ising model, as the Demon[3] and the Heat-Bath[4] algorithms. In this work we refer to the Heat-Bath algorithm and we show how it can be efficiently implemented on an FPGA-based dedicated architecture through the adoption of an High Level Synthesis design flow.
In the next Section B the HARWEST High Level Synthesis design flow is shortly reviewed; the Section C recalls the 3D-Ising Model and the Section D shortly reports about the Heat-Bath algorithm. Tha last Section E reports the results obtained when the HARWEST design flow is used to produce a dedicated architecture which implements the Heat-Bath algorithm on a Virtex4 Xilinx FPGA.

B The HARWEST High Level Synthesis Design Flow

The HARWEST High Level Synthesis (HLS) methodology, developed by Ylichron Srl, is one of the first outcomes of the HARWEST research project, aimed at creating a fully

automated HW/SW codesign environment. The HLS flow for the HW part is based on the flow reported in Fig. 1.
The system specifics are given by means of an ANSI C program (actually a subset of the ANSI C has been implemented, being pointers and recursion disallowed). At this very high level of abstraction, the correctness of the specifics is checked by running and debugging, on a conventional system, the C program (of course, we do not afford the task of proofing the correctness of a C program: we consider a program to be correct once it works correctly on some given sets of input data). Once the specifics have been fixed, they are translated into two different models of computation, namely the Control and Data Flow Graph (CDFG)[10] used to represent irregular, control dominated computations and the System of Affine Recurrence Equations (SARE)[7], used to represent regular, iterative computations characterized by affine dependencies on the loop nest indices. Given a set of resources (number of logic gates, number and sizes of memory banks, dimension of the I/O, ...), the two computational models are then mapped, with two distinct procedures, into a Data Path and a Control Finite State Machine (FSM) which enforce the behaviour expressed by the starting C program. Such an architecture, represented by the Data Path and the Control FSM, is further optimized (some redundand logic is removed and connections are implemented and optimized) and translated into an equivalent synthsizable VHDL code. Finally, the VHDL program is processed through the standard proprietary design tools (translation and map into the target technology, place and route) to produce the configuration bitstream.
It is worth to be underlined that the debugging phase is executed only at very high level of abstraction (on the C program), while all the other steps are fully automated and result to be correct by construction: this fact ensures that, once we have a working C program, the final architecture will show the same program behaviour.

B.1 Derivation of the CDFG and the SARE Computational Models

As reported in Fig. 1, the first step of the design flow has to transform the C program into an equivalent algorithm expressed by means of an instrinsically parallel computational model. In order to accomplish to this task, the HARWEST design flow uses two different modules to extract the CDFG and the SARE representation of the original C program. The sections of the C code that have to be transformed into the SARE formalism are explicitly marked by a pragma in the source code; if not otherwise specified, the C program is translated into the CDFG model.
When implementing the HARWEST flow, we decided to resort to a CDFG model with blocking semantics (i.e. a computing node does not terminate untill all its outputs have been read by the consumer nodes) because it does not require the insertion of buffers along the communication edges. As a consequence of the blocking semantics and of the precence of cycles on the CDFG, deadlocks could arise. In[13] we individuated a set of transformations into CDFGs of the basic C constructs (sequential statements, iteration, conditional) and of their compositions; these rules ensure deadlock never to arise. Thanks to these elementary transformations, each C program is transformed into an equivalent CDFG which does not deadlock.
The SARE model is used to deal with those portions of code

- constituted by the nesting of N iterative loops whose bounds are either constants or

affine functions of outer indices and

- having the statements in the loops that access m-dimensional arrays ($m \leq N$) through affine functions of the loop indices.

In order to extract the SARE representation of a C program which satisfies the previous two constraints, we have implemented the technique developed by P. Feautrier and described in[12].
An alternative way to express iterative affine computations is to use the SIMPLE language[8] which directly represents the SARE semantics since it allows to define

- the dimensionality of the computing space,
- the set of transformation equations with their affine dependencies and their definition domains,
- the set of input and output equations and their definition domains.

The HARWEST design flow accepts the definition of iterative affine computations through both C and SIMPLE programs.

B.2 Allocating and Scheduling CDFG and SARE

Resorting to the design flow in Fig. 1, after having derived the CDFG/SARE representation of the original algorithm, we are faced with the problem of allocating and scheduling such computations onto a set of predefined computing resources which represent a subset of the input constraints (further constraints could be involved by the need of a real-time behaviour or by throughput requirements).
In order to perform the allocation and scheduling operations, within the HARWEST design flow we created a library of computing modules which are the building blocks to set-up the final architecture. Each module is constituted by
• a collection of static informations regarding
▷ the area requirements (basic blocks needed for the implementation of the module onto a certain technology - LUT, memory modules, multipliers, ...),
▷ the module behaviour (combinational, pipelined, multicycle),
▷ the response time (propagation delay, setup time, latency, ...)
• a collection of files (EDIF format) to implement the module onto different target technologies (for instance, FPGAs from different vendors).
Allocating and scheduling a computation requires a time instant and a computing resource (i.e. a module) to be assigned to each operation of the computation. In order to do this, the HARWEST design flow implements a list scheduling heuristic, derived from the algorithm presented in[11], to deal with CDFG. It is worth to be underlined that the algorithm in[11] has been deeply modified to allow the sharing of the same HW module by different SW nodes.
When the original C program is structured with different functions, the program can be flattened through the inline expansion of the functions, thus generating a unique CDFG and a unique FSM. In order to reduce the complexity of the scheduling operation, as a simplifying option, each function - starting from the deepest ones - can be translated onto a different CDFG. The HARWEST flow initially schedules the CDFGs corresponding to code which

has not inner functions; successively such FSMs are scheduled as asynchronous modules when the CDFG of the calling function is scheduled; these asynchronous modules are synchronized with the FSM of the calling CDFG through a basic start/stop protocol.

Regarding the SARE computations, once the iterative N-dimensional algorithm has been expressed in the SARE formalism, it is allocated and scheduled onto a virtual parallel architecture - with a given spatial dimensionality $p < N$ - by means of an affine space-time transformation which projects the original algorithm, represented through some affine dependencies on the N-dimensional integer lattice, into the p-dimensional processor space[9],[7]; the remaining $N - p$ dimensions are allocated to the temporal coordinate and are used to fix the schedule. According to the input constraints on the architecture size, a clustering step is eventually performed to allow the final architecture to fit into the available resources.

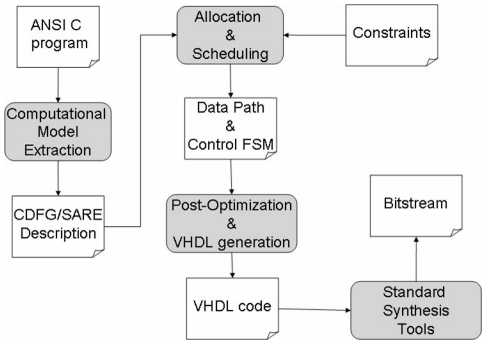

Figure 1. The HARWEST High Level Synthesis design flow

B.3 Post-Optimization and VHDL Generation

Once obtained the allocation and the scheduling for the CDFG and/or the SARE, the final architecture has still to be defined in the part concerning the interconnection/control network. In fact, during the scheduling step, all the details about the connections and the structures to support HW sharing are neglected. In this final phase all the connections are implemented, introducing and defining the multiplexing logic necessary to implement the communication lines; furthermore, a set of registers and multiplexers on the control lines are inserted, in order to allow a coherent management of iterations in presence of HW sharing.

After previous two steps have been executed, all the informations regarding the final architecture are fixed and the synthesizable VHDL code can be generated.

C The 3D Ising Model

Let us refer to the 3D Edwards-Anderson spin-glass model[5], where a variable σ_i ($i = \pm 1$) is present in each point of a L sized cubic lattice. A variable in site i is coupled to a variable in a neighbouring site j through the (static) random coupling factor $F_{ij} = \{\pm 1\}$. The system energy is given by

$$U = \sum_{i \in Lattice} \left(\sum_{j \in Neigh(i)} F_{ij} \sigma_i \sigma_j \right) \tag{C.1}$$

being the first summation running over all the lattice sites and the second spanning the neighbourhood of each lattice site. In order to study the system properties, a lot of Monte Carlo runs have to be executed, averaging over different choices of the (random and static) coupling factors. This problem is such a computational challenge that Italian and Spanish physicists are running a joint project to set-up a dedicated FPGA-based parallel computer (Ianus) to purposely address the aforementioned computation[6]. In our opinion, the main disadvantage of a Ianus-like approach to special purpose computing is that the architecture has to be manually developed, through the slow and error prone classical hardware design flow. On the base of our experience, we think that FPGA based computing architectures should be coupled with the - now emerging - HLS methodologies.

D The Heat Bath Algorithm

The kernel to implement the Heat-Bath algorithm, expressed in a C-like language, is shown through the following portion of code.

```
int spin[L][L][L];
int Jx[L][L][L];
int Jy[L][L][L];
int Jz[L][L][L];
for (k =0; k <L; k++ )
  for (j =0; j <L; j++ )
   for (i=0; i <L; i++ ) {
    nbs = spin[(i+1)modL][j][k]*Jx[(i+1)modL][j][k]+
     spin[(i-1) mod L][j][k]*Jx[(i-1)modL][j][k]+
     spin[i][(j+1)modL][k]*Jy[i][(j+1)modL][k]+
     spin[i][(j-1)modL][k]*Jy[i][(j-1)modL][k]+
     spin[i][j][(k+1)modL]*Jz[i][j][(k+1)modL]+
     spin[i][j][(k-1)modL]*Jz[i][j][(k-1)modL];
    if ( rand () < HBT( nbs) )
     spin[i][j][k] = +1;
    else
     spin[i][j][k] = -1;
   }
```

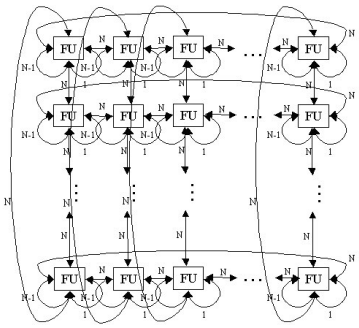

Figure 2. Structure of the parallel architecture produced by the HARWEST Design flow applied to the SARE expression of the Heat Bat algorithm

Thanks to its regularity, its transformation in the SARE formalism is straightforward. For this reasons, and because the efficiency of manually written SARE is usually higher with respect to the automatically generated ones, we decided to directly implement the Heat-Bath algorithm through the SIMPLE language.
In the previous code, the 3D array spin[L][L][L] contains the status (i.e. spin up or spin $down$) of all the lattice sites. The Jx, Jy and Jz arrays are used to store the (static) random coupling factors in the three coordinate directions (they represents the F_{ij} factors in Eq. C.1). The three nested for cycles perform the spin update in all the lattice sites. The 3D system spin dynamic is simulated by iterating over time the previous three loops.

E Results

After applying the design flow depicted in Fig. 1 to the SARE algorithm derived from the Heat-Bat code sketched above, we obtained the parallel architecture - constituted by $N \times \frac{N}{2}$ Functional Units - reported in Fig. 2 (N=L=24). Edges represent communication paths and the labels are the time delay (in clock cycles) associated to each edge. The picture of the architecture is reported to show how the regularity of the initial algorithm is maintained in the final architecture.

Each FU is constituted by

- One Random Number Generator, implemented through a 32-bit shift register feedback algorithm;
- One LUT (HBT[7] in the Heat-Bath code (Section D)) containing the probability values to decide wether a spin value should be set or reset, according to a comparison with the locally generated random value;
- One computing engine which, starting from the local spin value, the neighbouring spin values and the coupling coefficients, computes the "local energy" value to address HBT

- One update engine which, on the basis of HBT(local energy) and the local random value, determines the new spin value.

In our tests we set L=24 and, on the Xilinx Virtex4 FPGA XC4VLX160, we obtained the following synthesis results:

- Number of Slices: 61918
- Number of Slice Flip Flops: 95906
- Number of 4 input LUTs: 57678
- Number of FIFO16/RAMB16s: 168
- Number used as RAMB16s: 168
- f_{ck} = 91.5 MHz

According to the scheduling enforced by the HLS flow, 2×24 clock cycles are necessary to update the whole cubic lattice, corresponding - at the synthesis clock frequency of 91.5 MHz - to 520 ns. Resorting to the time necessary to update a single spin site (a commonly used reference figure), the synthesized architecture is able to update a spin location every 38 ps. To give an idea of the quality of such a result, we report in Table 1 the time extimations, derived in[6], for the time necessary to update a spin using optimized SW implementations on a Blue Gene/L node, on a Pentium4 processor, on a Cell processor and on the ClearSpeed CSX600 processor:

Processor	Spin Update Time (ps)	SlowDown vs FPGA
Blue Gene/L	45000	1184
Pentium 4	24000	632
Cell	1250	33
CSX600	1330	35

Table 1. comparative results for the implementation of the Heat Bath algorithm on different processor architectures

F Conclusions

In this work we addressed the feasibility of an High Level Synthesis (HLS) approach to design a FPGA-based architecture dedicated to the simulation of the 3D Ising model. To this aim, we used the HARWEST HLS design flow developed by Ylichron S.r.l., achieving - in about two week - a working prototype which offers up to three order of magnitudo speed-up with respect to optimized SW implementations on current high-end processing nodes. These results clearly demonstrate how the automatic exploitation of low level parallelism within FPGA devices characterized by very high internal memory bandwidth could be a very efficient answer to some classes of computationally challenging problems. Furthermore, we think that the very good results achieved (in terms of performances and of

developing time), make reasonable the adoption of FPGA based computing architectures which can be reliably, quickly and efficiently programmed through the HLS developing environments such as the one presented in this work.

Acknowledgements

We are indebted to Raffaele Tripiccione and Filippo Mantovani which introduced us to the Ising 3D model.

References

1. S. G. Brush, *History of the Lenz-Ising model*, Reviews of Modern Physics, **39**, 883–893, (1967).
2. B. A. Cipra, *The Ising model is NP-complete*, SIAM News, **33**, 6.
3. J. J. Ruiz-Lorenzo and C. L. Ullod, *Study of a microcanonical algorithm on the J spin-glass model in d=3*, Comp. Physics Communications, **105**, 210–220, (2000).
4. M. Creutz, *Quantum Fields on the Computer*, (World Scientific, 1992).
5. M. Mezard, G. Parisi and M. A. Virasoro, *Spin Glass Theory and Beyond*, (World Scientific, 1997).
6. F. Belletti et al., *Ianus: an adaptive FPGA computer*, in: IEEE Computing in Science and Engineering, **1**, 41–49, (2006).
7. A. Marongiu and P. Palazzari, *Automatic implementation of affine iterative algorithms: design flow and communication synthesis*, Comp. Physics Commun., **139**, 109–131, (2001).
8. A. Marongiu, et al., *High level software synthesis of affine iterative algorithms onto parallel architectures*, in: Proc. 8th International Conference on High Performance Computing and Networking Europe (HPCN Europe 2000), May 8–10, Amsterdam, (2000).
9. A. Marongiu and P. Palazzari, *Automatic mapping of system of N-dimensional affine recurrence equations (SARE) onto distributed memory parallel systems*, IEEE Trans. Software Engineering, **26**, 262–275, (2000).
10. G. G. De Jong, *Data Flow Graphs: System specification with the most unrestricted semantics*, in: Proc. EDAC, Amsterdam, (1991).
11. G. Lakshminarayana et al., *Wavesched: a novel scheduling technique for control flow intensive designs*, IEEE Trans on CAD, **18**, 5, (1999).
12. P. Feautrier, *Data flow analysis of array and scalar references*, Int. J. of Parallel Programming, **20**, 23–52, (1991).
13. G. Brusco, *Generazione di control data flow graph a partire da linguaggi imperativi e loro schedulazione*, Master Thesis in Electronic Engineering, University "La Sapienza" (Rome), (2004, in Italian).